3.6 `if __name__ == '__main__'`

One small application of if statements that we've already taken for granted in this course is writing certain "boilerplate" code for running certain libraries on our file. For example, we saw in 2.6 Testing Functions I: `doctest` and `pytest` that we add the following code to our Python file to run the doctests in that file:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Now that we've learned about if statements, we are ready to understand that first line, `if __name__ == '__main__'`.

import statements revisited

In 2.4 Importing Modules, we learned that an import statement is an instruction to the Python interpreter to find a Python module with a specified name and run it. This allows the program that executes the import statement to access the functions and data types defined within that module.

One consequence of this behaviour, though, is that by default *all* statements in the imported module are executed, not just function and data type definitions.

For example, suppose we had the following file `useful.py`:

```
# useful.py

def useful_function1(x: int) -> int:
    """...

    >>> useful_function1(1)
    110
    """
    # Body omitted


def useful_function2(s: str) -> str:
    """...

    >>> useful_function1('Hello')
    'Hello David'
    """
```

```
    # Body omitted


import doctest
doctest.testmod()
```

Note that here, the code to run `doctest` is not indented inside an if statement. It turns out that we can still run this file in the Python console, and the doctests will be run. However, these statements will also be executed every time `useful.py` is imported by another Python program. In other words, any time another program writes `import useful`, the doctests inside `useful.py` will be run, even though the doctests are not relevant for a program that just wants to use `useful.py`!

Enter `__name__`

To fix this problem, the Python interpreter creates a special variable called `__name__` for each module when a program is run.[1] By default, the `__name__` variable is set to the name of the module: the `__name__` of `useful.py` is `'useful'`, and the `__name__` attribute of `math` is `'math'`.

[1] Python uses the "double underscore" naming convention to denote special variable or function names. We'll encounter a few more of these throughout the course.

```
>>> import math
>>> math.__name__
'math'
```

However, when you *run* a module (e.g., right-click and select "Run File in Python Console"), the Python interpreter overrides the default module `__name__` and instead sets it to the special string `'__main__'`. And so checking the `__name__` variable is a way to determine if the current module is being run, or whether it's being imported by another module!

When we write `if __name__ == '__main__'`, we are really saying, "Execute the following code if this module is being run, and ignore the following code if this module is being imported by another module". The boolean expression `__name__ == '__main__'` evaluates to `True` in the former case, and `False` in the latter, and the conditional execution happens because of the if statement.

Organizing a Python file

We call this if branch (all the code under `if __name__ == '__main__'`) the module's **main block**. Here are some important conventions to follow for organizing a Python file with a main block:

1. The only code that goes outside of the main block are import statements (`import ...`), constant definitions (`MY_CONSTANT = ...`), function definitions (`def ...`), and data type definitions (`class ...`), which we will see in the next chapter.
2. Other code, like code for running `doctest` or `pytest`, goes inside the main block so that it is only executed when the module is run, and not when it is imported.
3. The main block goes at the bottom of the module.